

Lecture 9: *Online Algorithms for Metrical Task Systems
& Dating Problems*
Advanced Algorithms

Sid Chi-Kin Chau

Australian National University

✉ sid.chau@anu.edu.au

October 6, 2022

What is Love?

ONE DAY, PLATO ASKED SOCRATES, HIS TEACHER AND MENTOR, "What is love?"

SOCRATES REPLIED, "Plato, take a walk through the wheat field nearby. Without turning back, walk forward, and pick the most magnificent stalk of wheat you can find. However, you are allowed to pick only one."

PLATO FOLLOWED SOCRATES' INSTRUCTIONS, CONFIDENT THAT HE WOULD FIND THE BEST STALK OF WHEAT IN THE FIELD. BEFORE LONG THOUGH, HE RETURNED EMPTY-HANDED.

SOCRATES ASKED, "Why have you picked nothing?"

PLATO REPLIED, "I had found the most magnificent stalk of wheat as soon as I walked into the field, but since I was only allowed one pick, and I could not turn back, I thought I could find a better one further ahead. However, I could not find a better one as I kept searching, so I returned with none."

"And that is love," SAID SOCRATES.

《《 Tightrope Walking influenced by Adversary 》》

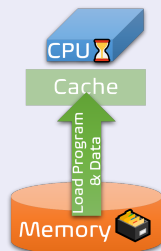


Caching Problem

- von Neumann computer model:
 - ▶ CPU needs to load the data blocks to cache before execution
 - ▶ Direct execution from memory will be very slow

Definition (Caching Problem (aka Paging Problem))

- Consider a memory with N data blocks (or called pages)
- There is a cache with space for only $K (\ll N)$ blocks
 - ▶ Some blocks are pre-loaded to the cache to speed up the access time
 - ▶ When the cache is full, we have to discard some loaded blocks
 - ▶ When a block is not missing in the cache when it is requested, a cache miss will occur
- Goal: Online algorithm to minimize the number of cache misses without knowing the future requested blocks
 - ▶ Decide which loaded blocks to be discarded when loading a new block



Online Caching Algorithm

- There are many online caching algorithms:
 - ▶ **FIFO** (first-in-first-out) removes the oldest block in cache
 - ▶ **LIFO** (last-in-first-out) removes the newest block in cache
 - ▶ **FWF** (flush-when-full) completely empties the cache when the cache is full and there is a cache miss
 - ▶ **LFD** (longest-forwarded-distance) removes the block that will be requested the latest by guessing the future requests
 - ▶ **LFU** (last-frequently used) removes the block that was requested least of them
 - ▶ **LRU** (least-recently used) removes the block requested least recently
- Which online caching algorithm is the best?
- Offline optimal solution:
 - ▶ Discard the block whose next request is farthest in the future

Online Caching Algorithm

Online Algorithm: \mathcal{A}_{LRU} (Least Recently Update)

- If cache is full, then discard the least recently requested block
- \mathcal{A}_{LRU} may incur a cache miss every time
 - ▶ E.g., consider $K = 3$ and request sequence $(1,2,3,4,1,2,3,4,1,2,3,4, \dots)$
- But offline optimal incurs a cache miss every K items
- The competitive ratio of \mathcal{A}_{LRU} is K

Lemma

No deterministic online algorithm can have a competitive ratio lower than K

Proof:

- Suppose $N = K + 1$. Adversary can observe which blocks discarded by online algorithm
- Adversary always requests the one block not in cache \Rightarrow A cache miss on every request
- Opt has only one cache miss every K requests – discard the next $(K + 1)$ -th request block

Online Caching Algorithm

Randomized Online Algorithm $\mathcal{A}_{\text{rcache}}$

- Start with all blocks unmarked, K arbitrary blocks in cache
 - When a block is requested
 - ▶ If it is in cache already, then mark it
 - ▶ If it is not and cache is full,
 - ★ Discard a random unmarked block
 - ★ Load the requested block into the cache, and mark it
 - If all blocks in cache are marked, unmark everything first
-
- **Oblivious Adversary** knows the online algorithm, but does not know what the random decisions are made by the online algorithm
 - ▶ We assume that oblivious Adversary prepares all the input in advance without knowing which random blocks will be discarded from the cache by the online algorithm
 - ▶ Oblivious Adversary is sufficient to model the adversarial nature in practice

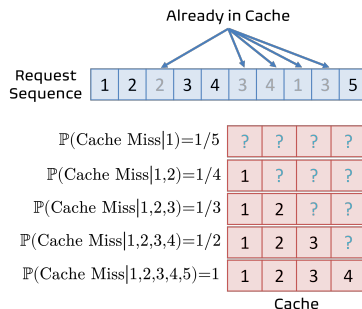
Online Caching Algorithm

Theorem

The expected competitive ratio of $\mathcal{A}_{\text{rcache}}$ is $O(\log K)$

Proof:

- Suppose the total number of blocks is $N = K + 1$
- Consider an iteration with a request sequence of $K + 1$ different blocks
 - ▶ Ignore the blocks in the requests that are requested for the second or more times, as they will not have cache miss
- Opt has at least one cache miss in the iteration



Online Caching Algorithm

Proof (Cont.):

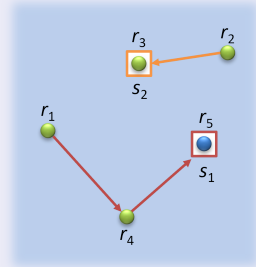
- In $\mathcal{A}_{\text{rcache}}$, the block not in cache must be a *random* unmarked block
- When a block is requested, if it was already marked, then there is no cache miss
- If it is not marked, then the probability $\mathbb{P}(\text{cache miss}) = \frac{1}{i}$, where i is the number of unmarked blocks
- By linearity of expectation, the expected total number of cache misses in the iteration is

$$\mathbb{E}[\text{num. of cache miss}] = \sum_{i=1}^N \mathbb{P}(\text{cache miss}) = \frac{1}{K+1} + \frac{1}{K} + \dots + \frac{1}{2} + 1 = O(\log K)$$

k -Server Problem

Definition (k -Server Problem)

- Let \mathcal{M} be a metric space with a distance function $d(\cdot, \cdot)$
 - ▶ $d(x, y)$ is a function to measure the distance between $x, y \in \mathcal{M}$
 - ▶ E.g., Euclidean space is a metric space
- There are k servers $\{s_1, \dots, s_k\}$
- R is a sequence of n requests (r_1, r_2, \dots, r_n) , such that $r_t \in \mathcal{M}$, which must be served by one of the k servers
 - ▶ If a new request r_t is not occupied by a server, we move a server, say s_j originally located at $r \in \mathcal{M}$, to r_t at a cost of $d(r, r_t)$
- Goal: Online algorithm to minimize the total travel distance by the servers, without knowing the future requests
 - ▶ Decide which server to be moved to the new request's location



k -Server Problem

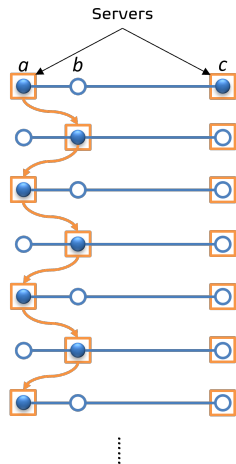
- Special cases of k -server problem
 - ▶ Caching problem
 - ★ \mathcal{M} = a set of blocks, $d(\cdot, \cdot) = 1$, locations of k servers = k blocks in cache
 - ▶ k -Headed disk problem
 - ★ $\mathcal{M} = [0, 1]$, $d(x, y) = |x - y|$
- Optimal Offline Algorithm:
 - ▶ Reduce to minimum-cost flow problem by dynamic programming

Definition (Lazy Algorithm)

- A lazy algorithm for k -server problem only moves servers when there is no server at the location of a request
- A lazy algorithm moves one server and only to the location of a request

Greedy Algorithm for k -Server Problem

- Greedy algorithm:
 - ▶ Always move the closest server to serve the next request
- Consider three locations on a line (a, b, c) with two servers
 - ▶ Distances: $d(a, b) \ll d(b, c)$
 - ▶ Requests: $r_1 = a, r_2 = c, r_3 = b, r_4 = a, r_5 = b, r_6 = a, r_7 = b, \dots$
- Greedy algorithm will move the same server to serve locations a and b , because c is very far away
- Offline optimal solution:
 - ▶ Put two servers at a and b subsequently
- The competitive ratio of greedy algorithm $\rightarrow \infty$



Lower Bound for Deterministic Online Algorithm for k -Server Problem

Theorem

Let (\mathcal{M}, d) be a metric space, where $|\mathcal{M}| \geq k + 1$. There is no deterministic lazy online algorithm for k -server problem on \mathcal{M} with a competitive ratio $< k$

Proof:

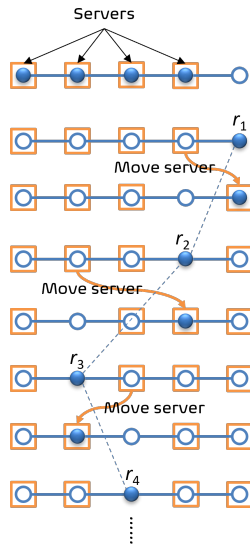
- Consider $B = \{b_1, \dots, b_{k+1}\} \subseteq \mathcal{M}$, a subset of \mathcal{M} with $k + 1$ locations
- We assume that online algorithm \mathcal{A} starts with servers at k different locations in B
- Algorithm \mathcal{A} is lazy and always moves at most one server to each location
- The adversary requests $R = (r_1, \dots, r_n)$ with locations in B at which \mathcal{A} has no server
- We show that the total distance by the servers in \mathcal{A} is lower bounded by

$$d_{\mathcal{A}}(R) \geq \sum_{t=1}^{n-1} d(r_t, r_{t+1})$$

Lower Bound for Deterministic Online Algorithm for k -Server Problem

Proof (Cont.):

- After r_{t-1} , the next r_t is requested at the location that was covered by the server before serving r_{t-1}
- Suppose a server is moved from some location p_t to serve r_t
- Note that the cost for servicing $d(p_t, r_t) = d(r_t, r_{t+1})$ for $t \leq n-1$
- Next, we show $\sum_{t=1}^{n-1} d(r_{t+1}, r_t) \geq k \cdot d_{\text{Opt}}(R)$, where Opt is offline optimal solution for R
- Consider k algorithms $\{\mathcal{A}_t\}$ for $t \leq k$ defined as follows:
 - ▶ Initially, all $\{\mathcal{A}_t\}$ starts by placing servers at $B \setminus \{b_{k+1}\}$
 - ▶ To service $r_1 = b_{k+1}$, algorithm \mathcal{A}_t moves the server from b_t to b_{k+1}
 - ▶ Observe that there exists exactly one location that is covered by all $\{\mathcal{A}_t\}$, i.e., b_{k+1}
 - ▶ Each algorithm \mathcal{A}_t is lazy and behaves so as to move a server between locations b_t and b_{k+1} only

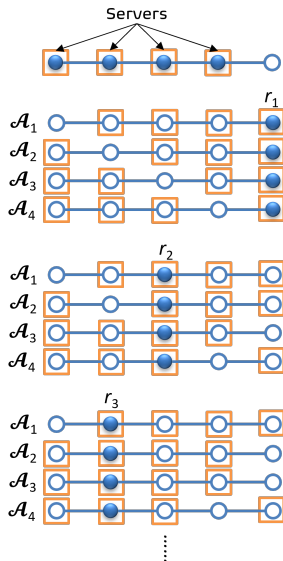


Lower Bound for Deterministic Online Algorithm for k -Server Problem

Proof (Cont.):

- Since we want to maintain that at all times there is exactly one location in \mathcal{M} that is covered by all algorithms $\{\mathcal{A}_t\}$, has to move the server that is presently at location b_{k+1}
- Let $\sum_{t=1}^{n-1} d_{\mathcal{A}_t}(R)$ denote the sum of costs of all algorithms $\{\mathcal{A}_t\}$. By induction we have that at each time step j , only one algorithm in $\{\mathcal{A}_t\}$ has to move a server and moreover it moves a server that is at location r_{j-1} . Thus, the total cost of all the \mathcal{A}_t is $\sum_{t=1}^{n-1} d(r_{t+1}, r_t)$
- In particular, there is one of the algorithms achieving a cost below the average cost, hence $d_{\text{Opt}}(R) \leq \frac{1}{k} \sum_{t=1}^{n-1} d(r_{t+1}, r_t)$,

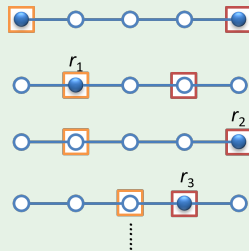
$$d_{\mathcal{A}}(R) \geq \sum_{t=1}^{n-1} d(r_{t+1}, r_t) \geq k \cdot d_{\text{Opt}}(R)$$



Double Covering Algorithm for k -Server Problem on a Line

Double Covering Algorithm (DC)

- Consider 1-dimensional setting on a line
- If a new request r_t is left (or right) of all servers, then move the leftmost (or right-most) server to r_t
- Otherwise, move the two servers left and right of r_t with the same velocity towards r_t . Stop both servers when one arrives at r_t
- Note that DC algorithm is not lazy
- But it will avoid the oscillation of greedy algorithm: far-away server will approach closer to the new request



Double Covering Algorithm for k -Server Problem on a Line

Theorem

Consider k -server problem on a line, the competitive ratio of double covering (DC) algorithm is k

Proof:

- We will prove that $\text{Cost}(\text{DC}) \leq k \cdot \text{Cost}(\text{Opt})$ using a potential function
- Let $p_1(t) \leq \dots \leq p_k(t)$ and $p_1^*(t) \leq \dots \leq p_k^*(t)$ be the servers' locations of DC and Opt respectively after the request r_t
- Define a potential function ϕ_t (note that $\phi_t \geq 0$) for the t -th step:

$$\phi_t = k \sum_{s=1}^t |p_s(t) - p_s^*(t)| + \sum_{s' < s} |p_{s'}(t) - p_s(t)|$$

- We write $\phi_t^1 = k \sum_{s=1}^t |p_s(t) - p_s^*(t)|$ and $\phi_t^2 = \sum_{s' < s} |p_{s'}(t) - p_s(t)|$

Double Covering Algorithm for k -Server Problem on a Line

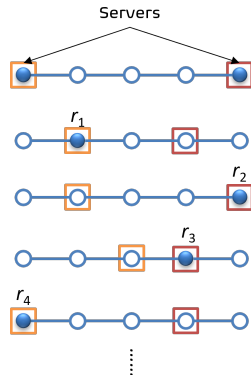
Proof (Cont.):

- Let $\Delta_t \phi = \phi_t - \phi_{t-1}$ be the change in ϕ at each request r_t , $\Delta_t \text{DC}$ be the cost by DC, and $\Delta_t \text{Opt}$ be the cost by Opt for request r_t
- To service r_t , both DC and Opt must move a server on r_t
- We evaluate $\Delta_t \phi = \Delta_t \phi^1 + \Delta_t \phi^2$ by thinking of Opt moving first and DC moving next

Move type	$\Delta_t \text{DC}$	$\Delta_t \text{Opt}$	$\Delta_t \phi^1$	$\Delta_t \phi^2$
Opt moves	0	$\Delta_t \text{Opt}$	$\leq k \Delta_t \text{Opt}$	0
DC moves (1 server)	$\Delta_t \text{DC}$	0	$-k \Delta_t \text{DC}$	$(k-1) \Delta_t \text{DC}$
DC moves (2 servers)	$\Delta_t \text{DC}$	0	≤ 0	$-\Delta_t \text{DC}$

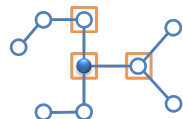
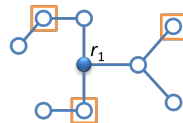
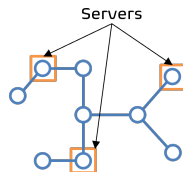
- Then $\text{Cost}(\text{DC}) \leq k \cdot \text{Cost}(\text{Opt})$ because $\phi_0 = 0$ and

$$0 \leq \phi_n - \phi_0 = \sum_{t=1}^n \Delta_t \phi \leq k \sum_{t=1}^n \Delta_t \text{Opt} - \sum_{t=1}^n \Delta_t \text{DC} = k \cdot \text{Cost}(\text{Opt}) - \text{Cost}(\text{DC})$$



Double Covering Algorithm for k -Server Problem on a Tree

- Let T be a tree embedded in the plane (also called tree metric space)
- For nodes x and y on tree T , the distance $d(x, y)$ is the Euclidean distance of the unique path between x and y
- A line can be viewed as a path on a tree and we want to extend the DC algorithm to a tree
- Server s is a **neighbor** of a request r if there is no other server on the path from s to r
- The servers that are neighbor to the new request move at the same speed toward the request
 - Exactly one server is a neighbor to the request – only one server is moving
 - Some $m \leq k$ servers are neighbors to the request – m servers are moving and $k - m$ servers are not moving
- The competitive ratio of DC algorithm is k for k -server problem on a tree



Probabilistic Embedding in Tree

Definition (Embedding in Metric Space)

- Let (\mathcal{M}, d) be a metric space. We say a metric space (\mathcal{M}', d') with $\mathcal{M} \leq \mathcal{M}'$ dominates \mathcal{M} if $d(x, y) \leq d'(x, y)$ for $x, y \in \mathcal{M}$. We call the ratio $\max_{x, y \in \mathcal{M}} \frac{d'(x, y)}{d(x, y)}$ the stretch in \mathcal{M}'
 - ▶ E.g., (\mathcal{M}, d) is a 2D Euclidean metric space, and (\mathcal{M}', d') is a tree metric space
- Let \mathcal{M} be a set of metrics that dominate \mathcal{M} , and P a probability distribution over \mathcal{M} . We say that (\mathcal{M}, P) is an α -approximation of \mathcal{M} , if $x, y \in \mathcal{M}$ and a random metric (\mathcal{M}', d') from \mathcal{M} according to probability distribution P :

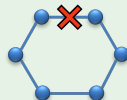
$$\mathbb{E}[d'(x, y)] \leq \alpha \cdot d(x, y)$$

- We also say that \mathcal{M} is embedded probabilistically in \mathcal{M}
- Can we find a probabilistic embedding of an Euclidean metric space in tree metric space with a low stretch?

Probabilistic Embedding in Tree

Example (Embedding in Tree)

- In the deterministic way there can not be an embedding better than $\Omega(n)$
 - ▶ E.g., consider a cycle with n nodes. Removing one edge gives a stretch of $n - 1$ for the worst case
- If we embed in a randomized manner, then the expected stretch will be
$$\sum_{i=1}^{n/2} \frac{1}{n} \cdot \frac{n-i}{i} + \frac{1}{n} \cdot \frac{n}{2} = \sum_{i=1}^{n/2} \frac{1}{i} = O(\log n)$$



Theorem (Fakcharoenphola-Rao-Talwar Theorem)

For a metric \mathcal{M} with n points, there is a set \mathcal{M} of tree metrics that dominate \mathcal{M} and probability distribution P over \mathcal{M} , subject to (\mathcal{M}, P) is a $O(\log n)$ -approximation of \mathcal{M}

- Probabilistic embedding in a tree metric space allows us to tackle k -server problem by applying double covering algorithm on a tree

Randomized Online Algorithm for k -Server Problem

Theorem

There is a randomized online algorithm for the k -server problem, which has an expected competitive ratio $O(k \cdot \log n)$ for any metric space with n locations

Proof:

- By Fakcharoenphola-Rao-Talwar Theorem, there is a probabilistic embedding in a tree metric space T with expected $O(\log n)$ -approximation
- Let $\text{Cost}(\cdot)$ and $\text{Cost}_T(\cdot)$ be the cost of a solution using metric distance functions $d(\cdot, \cdot)$ and $d_T(\cdot, \cdot)$ respectively
- Let DC_T and Opt_T be the solution of DC and optimal solution respectively on embedded tree metric space T :

$$\begin{aligned}\mathbb{E}[\text{Cost}(\text{DC}_T)] &= \mathbb{E}[\text{Cost}_T(\text{DC}_T)] \leq \mathbb{E}[k \cdot \text{Cost}_T(\text{Opt}_T)] \\ &\leq k \cdot \mathbb{E}[\text{Cost}_T(\text{Opt}_T)] \leq k \cdot O(\log n) \cdot \text{Cost}(\text{Opt})\end{aligned}$$

Metrical Tasks System Problem

- Metrical task systems (MTS) are an abstract model for general online decision problems
- MTS captures diverse problems, e.g. caching, k -server, uncertain exploration problems

Definition (Metrical Tasks System (MTS))

- MTS is a tuple (\mathcal{M}, S, R) where \mathcal{M} is a metric space with distance function $d(\cdot, \cdot)$
- $S \subseteq \mathcal{M}$ is a finite set in \mathcal{M} , representing a set of possible states
- R is a sequence of T requests (r_1, r_2, \dots, r_T) , and cost function $\text{Cost}(r_t, s)$ is the associated cost of processing request r_t at state $s \in S$
- Denote the current state at the t -th request by s_t
- Goal: Online algorithm to decide the state s_t for the t -th request, such that it will minimize the total cost (including the transition cost $d(s_{t-1}, s_t)$ and the processing cost $\text{Cost}(r_t, s_t)$), without knowing the future requests

Metrical Tasks System Problem

- Let $W_t[s]$ be a work function (i.e., lowest accumulative cost) at current state s :

$$W_t[s] = \min_{s' \in S} \left(W_{t-1}[s'] + \text{Cost}(r_t, s') + d(s', s) \right)$$

where $W_0[s] = 0$ for all s

- $W_t[s]$ can be computed offline by dynamic programming
- The cost of offline optimal solution: $\text{Cost}[\text{Opt}] = \min_{s \in S} W_n[s]$
- Let $\text{Opt} = (s_t^*)_{t=1}^n$ be an offline optimal solution, which is obtained by back-tracking
 - 1 Find s_n^* by

$$s_n^* = \arg \min_{s \in S} W_n[s]$$

- 2 Find $s_t^* = s$, given s_{t+1}^* , by finding a suitable s that satisfies

$$W_{t+1}[s_{t+1}^*] = W_t[s] + \text{Cost}(r_{t+1}, s_{t+1}^*) + d(s, s_{t+1}^*)$$

Work Function Algorithm

Work Function Algorithm (WFA) [Input: $(s_1, \dots, s_{t-1}), r_t$]

- At each request r_t , find s_{t-1} such that

$$s_t = \arg \min_{s \in S} \left(W_t[s] + d(s_{t-1}, s) \right)$$

subject to

$$W_t[s_t] = W_{t-1}[s_t] + \text{Cost}(r_t, s_t)$$

- Return s_t for the current state
- Basic idea: WFA finds a state that minimizes the discrepancy with the offline optimal cost for the current request, subject to the constraint that the state remains unchanged from the previous request in the least accumulative cost
- Note that there always exists $s \in S$ that satisfies $W_t[s] = W_{t-1}[s] + \text{Cost}(r_t, s)$. Otherwise, $W_t[s]$ cannot be the lowest accumulative cost

Work Function Algorithm

Theorem

If $|\mathcal{M}| = n$, the competitive ratio of work function algorithm is $(2n - 1)$ for MTS

Proof:

- Let $d_{\max} \triangleq \max_{s, s' \in S} d(s, s')$
- Define potential function $\phi_t = 2 \sum_{s \in S} W_t[s] - W_t[s_t]$, given the current selected state s_t
- We show that
 - ▶ Claim 1: $\phi_n \leq (2n - 1) \cdot \text{Cost}[\text{Opt}] + (2n - 2)d_{\max}$
 - ▶ Claim 2: $\phi_t - \phi_{t-1} \geq \text{Cost}[\text{WFA}_t] - \text{Cost}[\text{WFA}_{t-1}]$, where WFA_{t-1} is the output for the $(t - 1)$ -th request
- By the two claims, we have

$$(2n - 1) \cdot \text{Cost}[\text{Opt}] + (2n - 2)d_{\max} \geq \phi_n = \sum_{t=1}^n (\phi_t - \phi_{t-1}) \geq \text{Cost}[\text{WFA}_n]$$

Work Function Algorithm

Proof (Cont.):

- To prove Claim 1, note that $W_t[s] \leq \min_{s' \in S} W_t[s'] + d_{\max}$ for all $s \in S$

$$\begin{aligned}\phi_n &= W_n[s_n] + 2 \sum_{s \neq s_n} W_n[s] \leq W_n[s_n] + 2(n-1) \left(\min_{s \in S} W_n[s] + d_{\max} \right) \\ &\leq (2n-1) \cdot \min_{s \in S} W_n[s] + (2n-2)d_{\max} = (2n-1) \cdot \text{Cost}[\text{Opt}] + (2n-2)d_{\max}\end{aligned}$$

- To prove Claim 2, note that WFA selects s_t such that

$$W_t[s_t] + d(s_{t-1}, s_t) = \min_{s \in S} \left(W_t[s] + d(s_{t-1}, s) \right) \Rightarrow W_t[s_t] + d(s_{t-1}, s_t) \leq W_t[s_{t-1}]$$

- Also, WFA satisfies $W_t[s_t] - W_{t-1}[s_t] = \text{Cost}(r_t, s_t)$. Hence,

$$W_t[s_t] + d(s_{t-1}, s_t) + \text{Cost}(r_t, s_t) \leq W_t[s_{t-1}] + W_t[s_t] - W_{t-1}[s_t]$$

Work Function Algorithm

Proof (Cont.):

- To prove Claim 2, we have



$$\text{Cost}[\text{WFA}_t] - \text{Cost}[\text{WFA}_{t-1}] = d(s_{t-1}, s_t) + \text{Cost}(r_t, s_t) \leq W_t[s_{t-1}] - W_{t-1}[s_t]$$

- From the definition of ϕ_t

$$\begin{aligned}\phi_t - \phi_{t-1} &= 2\left(\sum_{s \in S} W_t[s] - \sum_{s \in S} W_{t-1}[s]\right) - W_t[s_t] + W_{t-1}[s_{t-1}] \\ &= 2\left(\sum_{s \neq s_t \wedge s \neq s_{t-1}} W_t[s] - W_{t-1}[s]\right) + W_t[s_{t-1}] - W_{t-1}[s_t] \\ &\geq W_t[s_{t-1}] - W_{t-1}[s_t] \geq \text{Cost}[\text{WFA}_t] - \text{Cost}[\text{WFA}_{t-1}]\end{aligned}$$

Because $W_t[s] - W_{t-1}[s] \geq 0$

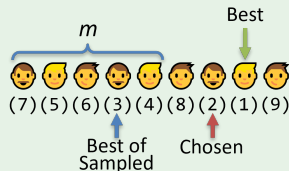
Online Dating Problem

- Goal: Find your true love by dating a stream of candidates
- Suppose you estimate that you will have n dates over the time
 - ▶ Your true love will be the best person out of the n candidates
- Rules:
 - ▶ You can only date one person at a time (no cheating allowed )
 - ▶ You have to decide whether either you will
 - ★ Marry the current candidate
 - ★ Or break up with the current candidate to date the next (unknown) candidate
 - ▶ Broken-up relationship can't be rekindled (need to move on from past relationships )
- Dating is definitely an “online” decision problem
- Also known as secretary problem (for hiring a secretary from a stream of candidates)

Online Dating Problem (or Secretary Problem)

Sample-and-Choose Algorithm

- Evaluate the first m candidates, and rank them
- Reject all the first m candidates
- After the first m candidates, choose the one who is at least as good as the best in the first m candidates, or the last one



Theorem

The probability of choosing the best candidate by sample-and-choose algorithm is maximized to be $\frac{1}{e}$, when $m = \frac{n}{e}$ (also known as the “37-percent rule”)

Proof:

- Let the probability of choosing the best candidate by sample-and-choose be $P_n(m)$
- $P_n(m) = \sum_{i=m+1}^n \mathbb{P}(i\text{-th candidate is the best} \wedge i\text{-th candidate is chosen})$

Online Dating Problem (or Secretary Problem)

Proof (Cont.):

- If the i -th candidate is chosen, then the best of the previous $(i - 1)$ candidates must be in the first m candidates. Otherwise, one of the $(i - 1 - m)$ candidates will be chosen
- Then $\mathbb{P}(i\text{-th candidate is the best}) = \frac{1}{n}$ and $\mathbb{P}(i\text{-th candidate is chosen}) = \frac{m}{i-1}$,

$$P_n(m) = \frac{1}{n} \left(1 + \frac{m}{m+1} + \frac{m}{m+2} + \dots + \frac{m}{n-1} \right) = \frac{m}{n} \sum_{i=m}^{n-1} \frac{1}{i} \approx \frac{m}{n} \ln\left(\frac{n}{m}\right)$$

- Note that

$$\frac{d(x \ln(1/x))}{dx} = -1 + \ln(1/x)$$

The maximum point is $x^* = \frac{1}{e}$

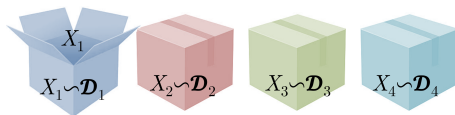
- Hence, $P_n(m)$ is maximized, when we set $\frac{m^*}{n} = \frac{1}{e}$, and $P_n(m^*) = \frac{1}{e}$

Online Dating Problem (or Secretary Problem)

- Maximizing the probability of choosing the best candidate may not be the best objective
 - ▶ Sample-and-choose algorithm either selects the best, or the last appearing candidate
 - ▶ With constant probability, the rank of the last candidate is $\Omega(n)$
- How about maximizing the expected rank of the selected candidate?
 - ▶ Still an open problem, no known solution (as known as Robbin's problem of optimal stopping)
 - ▶ Online dating is an unsolved hard problem 😂
- Generally, optimal stopping problem:
 - ▶ Optimal stopping problem aims to choose a time to take a particular action, given a sequence of input revealed sequentially, in order to maximize an expected reward or minimize an expected cost
 - ▶ Applications: operations research, economics, and mathematical finance

Prophet Inequality

- Consider a sequence of random variables $(X_i)_{i=1}^n$ arrive from known distributions \mathcal{D}_i
- When each X_i arrives, the decision-making process must decide whether to accept it and stop the process, or whether to reject it and go on to the next variable in the sequence



Definition (Prophet Inequality)

- A prophet, knowing the whole sequence, can select the largest $X_{\max} \triangleq \max_{i=1}^n (X_i)$, for any instance of this process, and attain the expected value $\mathbb{E}[X_{\max}]$
- An online threshold strategy: Accept the revealed X_i , if $X_i \geq \theta$
- The prophet inequality states that the expected pay-off of threshold strategy $\geq \frac{1}{2} \mathbb{E}[X_{\max}]$, when setting θ such that $\mathbb{P}(X_{\max} \geq \theta) = \frac{1}{2}$

Prophet Inequality

Proof:

- Let $(X - \theta)^+ = \max(X - \theta, 0)$

$$\mathbb{E}[X_{\max}] \leq \theta + \mathbb{E}[(X_{\max} - \theta)^+] \leq \theta + \mathbb{E}\left[\sum_{i=1}^n (X_i - \theta)^+\right]$$

- The expected pay-off of online threshold strategy with threshold θ :

$$\begin{aligned} & \mathbb{E}[\text{Pay-off with threshold } \theta] \\ & \geq \theta \cdot \mathbb{P}(X_{\max} \geq \theta) + \sum_{i=1}^n \mathbb{E}[(X_i - \theta)^+] \cdot \mathbb{P}\left(\max_{j < i} X_j < \theta\right) \\ & \geq \theta \cdot \mathbb{P}(X_{\max} \geq \theta) + \sum_{i=1}^n \mathbb{E}[(X_i - \theta)^+] \cdot \mathbb{P}(X_{\max} < \theta) \end{aligned}$$

- Since $\mathbb{P}(X_{\max} \geq \theta) = \mathbb{P}(X_{\max} < \theta) = \frac{1}{2}$, $\mathbb{E}[\text{Pay-off with threshold } \theta] \geq \frac{1}{2}\mathbb{E}[X_{\max}]$

Reference Materials

- Online Computation and Competitive Analysis (Borodin, El-Yaniv), Cambridge Uni. Press
 - ▶ Chapters 3, 4, 8, 9, 10

Recommended Materials

- *Knowing When to Stop* (Hill), American Scientist, 2009